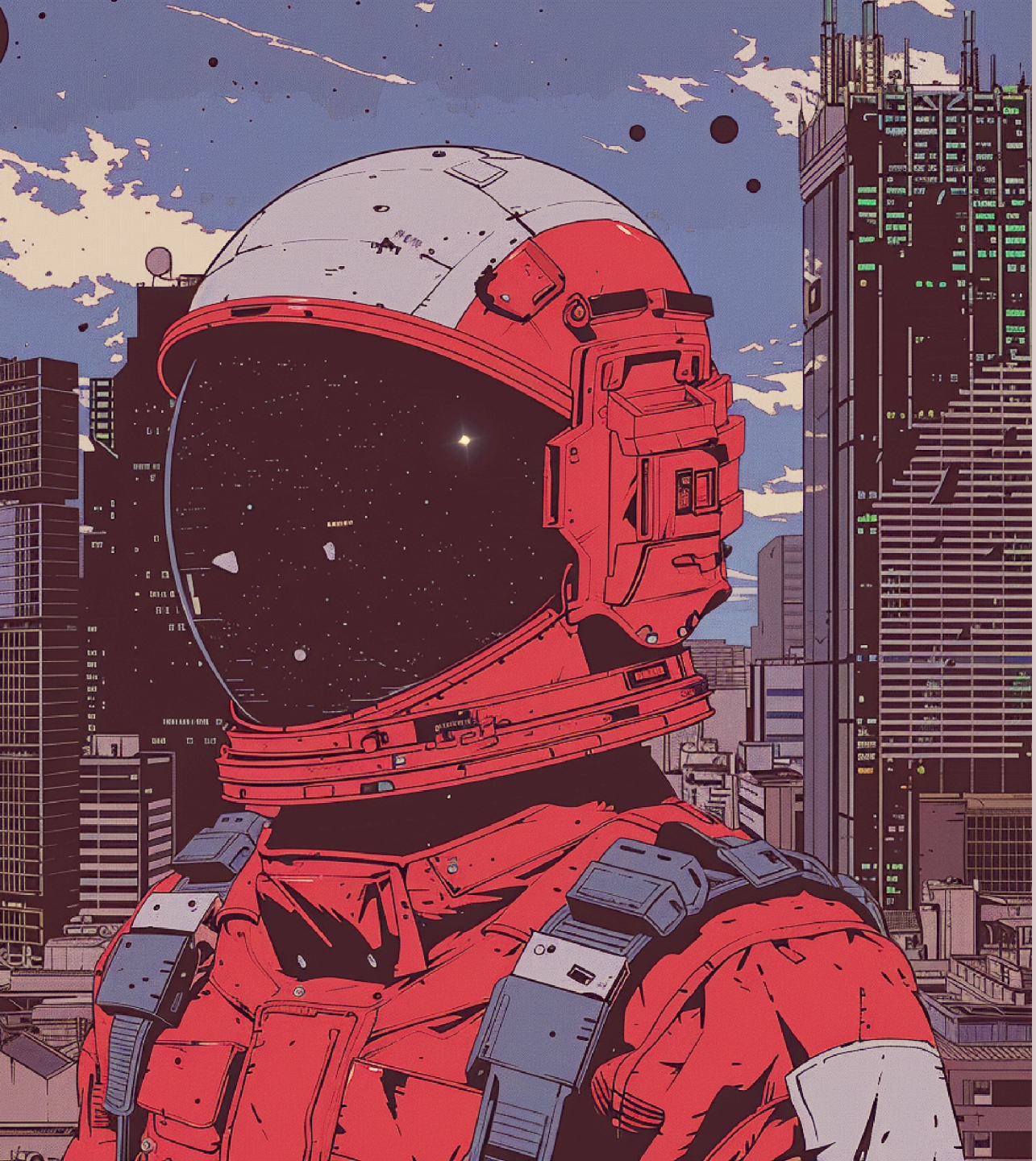


The Kernel Hacker's Guide to the Galaxy

Automating Exploit Engineering Workflows

H2HC 2024 São Paulo

IBM®

whoami (we)

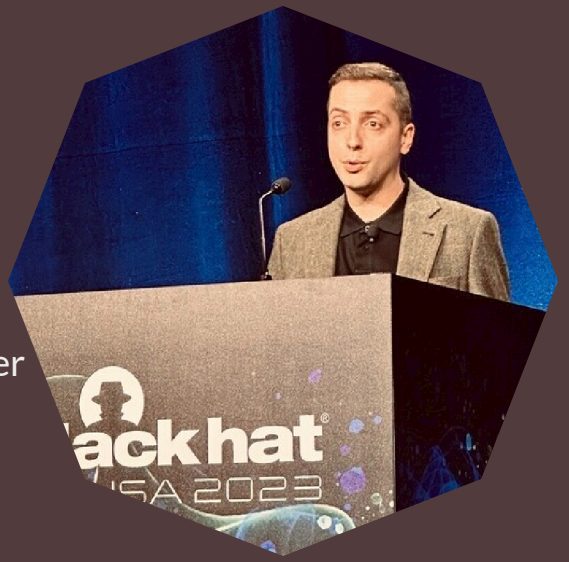
XOR

X-FORCE OFFENSIVE RESEARCH

chompie

weird machine mechanic

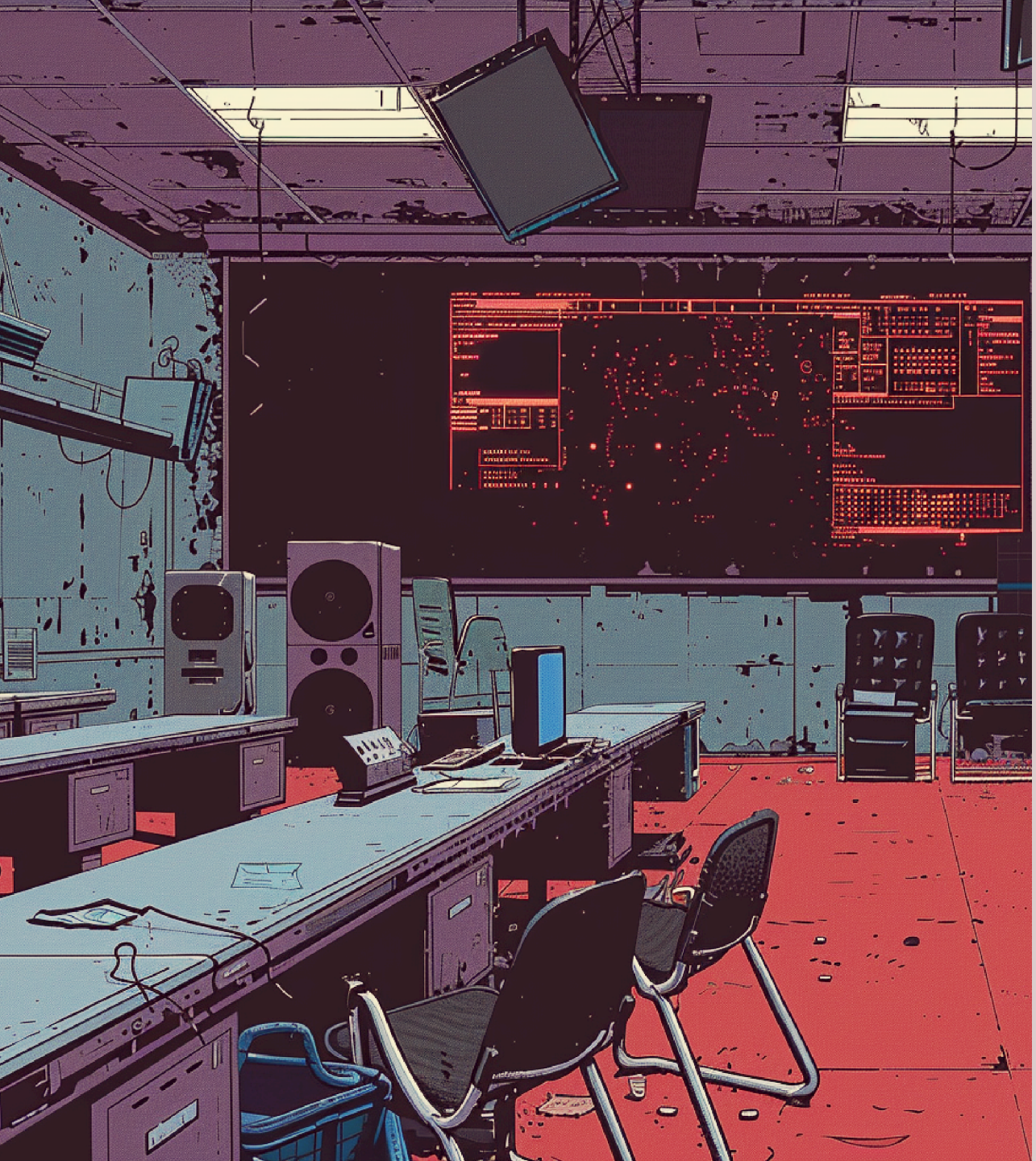
@chompie1337



FuzzySec

just-in-time developer

@FuzzySec



Introduction

Setting the Stage

Why are we talking about this?

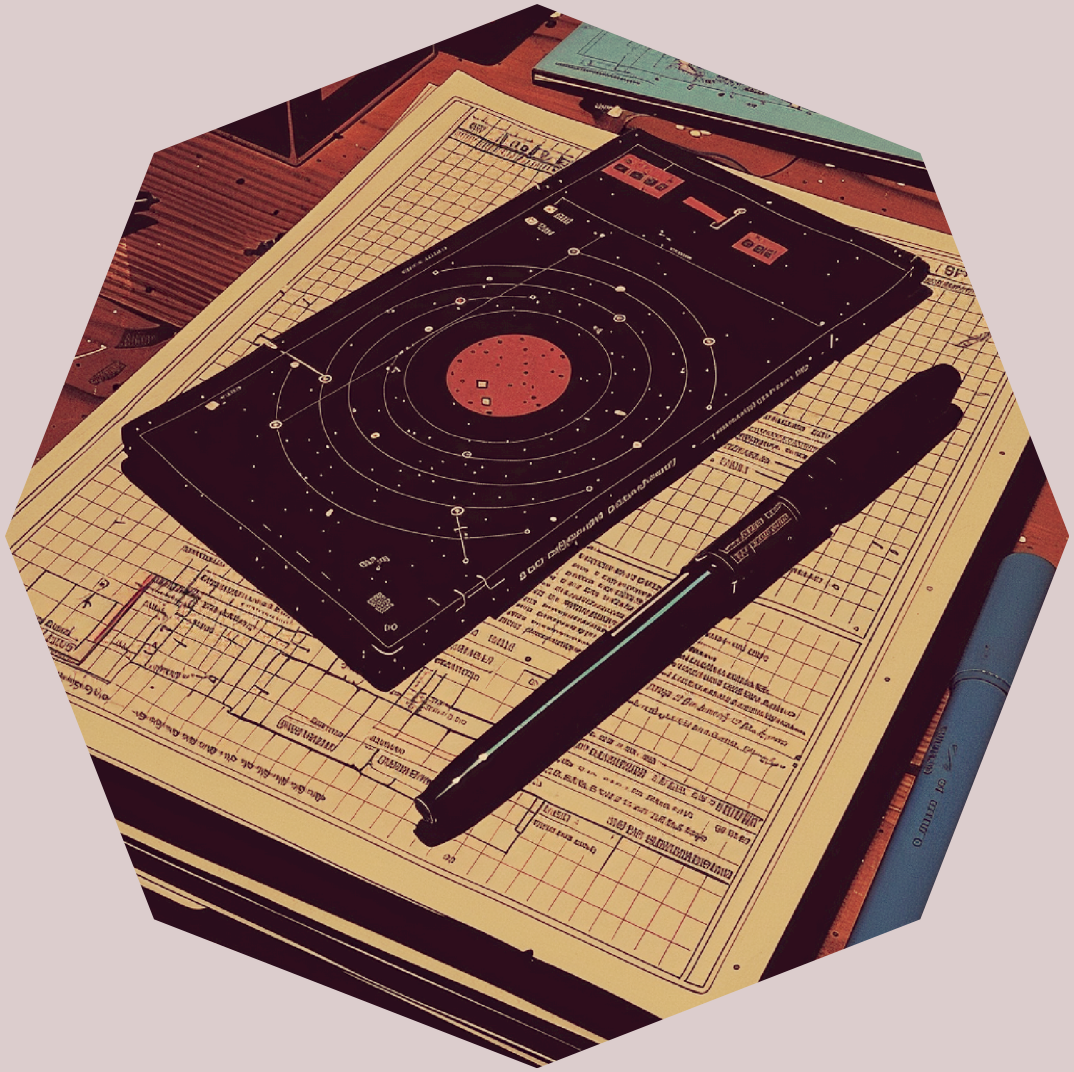
- **Practical business need**
  - N-Day **Patch Gaps**
  - **Sustain** our **sanity**
  - Reduce patch analysis time
  - **Approach a return** to Patch Tuesday ⇨ Exploit Wednesday
- **Modern mitigations and increased complexity**
  - Exploit construction is becoming **increasingly more labour intensive**
  - **Different targets** have differing considerations & **complexity** (e.g. mobile vs desktop vs IOT)

Exploits are code too 

There has been much discussion on **automatic vulnerability discovery**. What about **automating exploit building**, particularly for vulnerabilities that produce **non-trivial primitives**?

Past work

- Practical Uses of Program Analysis: Automatic Exploit Generation
  - Sophia D'Antoine
  - https://www.youtube.com/watch?v=d3fy4se4JO0
- The Path Towards Automated Heap Exploitation
  - Thaís Moreira Hamasaki
  - https://www.youtube.com/watch?v=5XQ0cAwlWMo
- Greybox Automatic Exploit Generation for Heap Overflows in Language Interpreters
  - Sean Heelan
  - https://www.cs.ox.ac.uk/tom.melham/phd/Heelan-2020-GAE.pdf
- Hot topic in 2016 Darpa Cyber Grand Challenge
  - Focus **shifted** now towards **AI vulnerability discovery**



Theory & Practice

- Past work is largely theoretical
- Focus on applying Automated Exploit Engineering (AEG) concepts to real targets
  - Windows Kernel LPE exploits
- Historical thought process
  - AEG will mostly apply to granular workflows
  - This aligns with our efforts

Theoretical AEG

Two Phases

- Code to trigger the vulnerability
- Generate payload from weird machine primitive

The Exploit Development Lifecycle (BSides Canberra Keynote 2024) by chompie

- https://drive.google.com/file/d/1jHnVdjAcPGkuVPiakZBAOTp8uzMej6LY/view

AEG in Practice

Many Phases

- Code to trigger the vulnerability
- Exploit primitive construction/transformation
- Mitigation bypasses
  - Information leaks, CFI gadgets, additive primitive contortion, avoidance of Kernel Patch Protection (KPP) and HVCI
- Stability Improvements
- Targeting
- Stealth

What is automation, really?

Immediate practical impact

- Self contained
- Atomic operations

Reduce manual labour

- Primitive discovery is an intensive task
- Suitable exploitation objects
- Control Flow gadgets

Maintain sanity

- Narrow search-time for broad deterministic questions
- Save brainpower for reverse engineering tasks that can't be automated (yet)

Maintenance & QA

- Does this work on this other targets?

Component Breakdown

1

Initial Trigger

- Generate user-space code to trigger the vulnerability

2

Exploitability analysis

- Code reachability
- Race condition analysis

3

Exploit primitive construction

- Heap spray object identification

4

Mitigation bypasses

- Primitive contortion
- kCFG gadgets

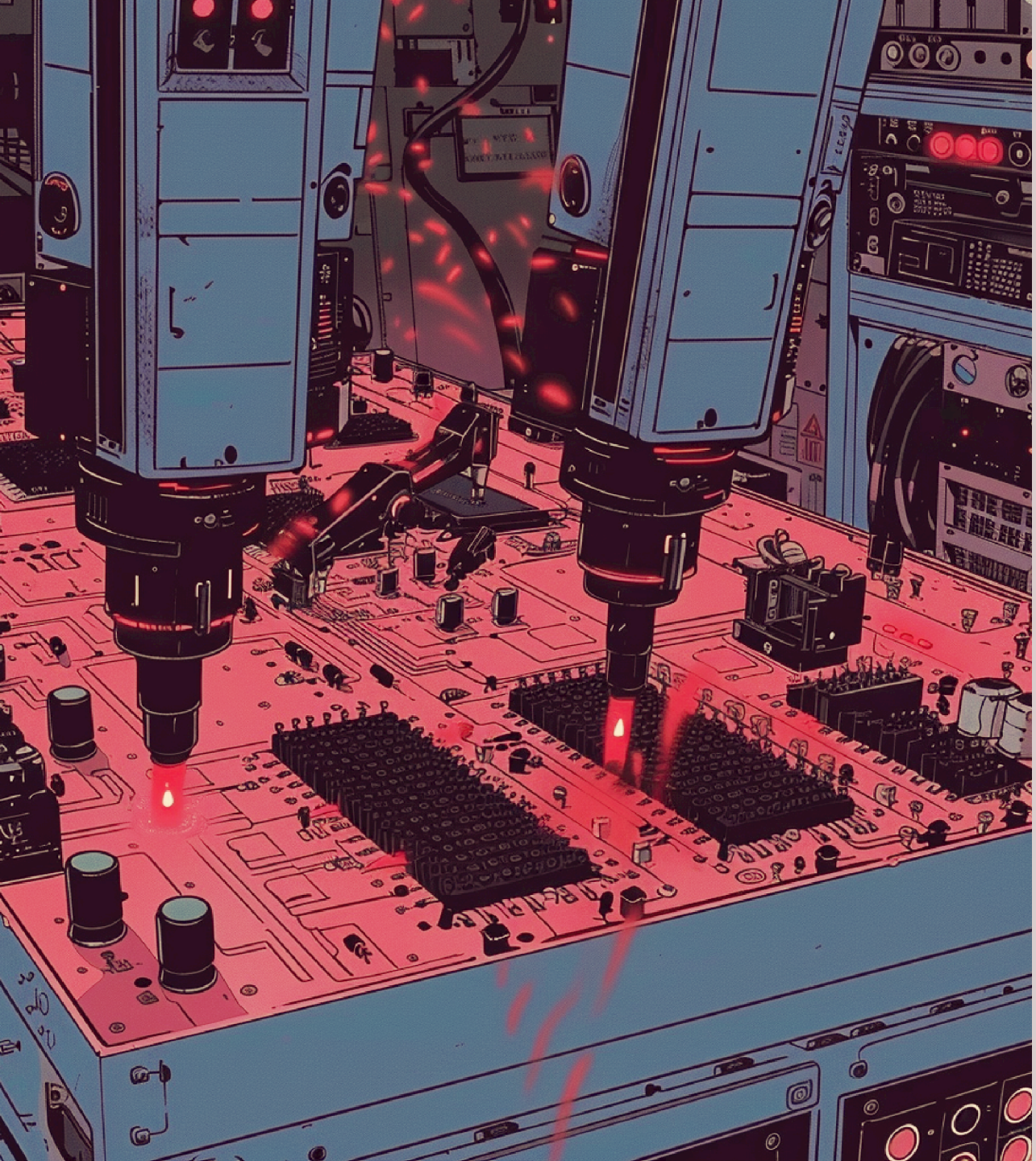
5

Stability Improvements

6

Variant analysis

- Does this pattern exist somewhere else?



The Atomics of Automation

Program Analysis

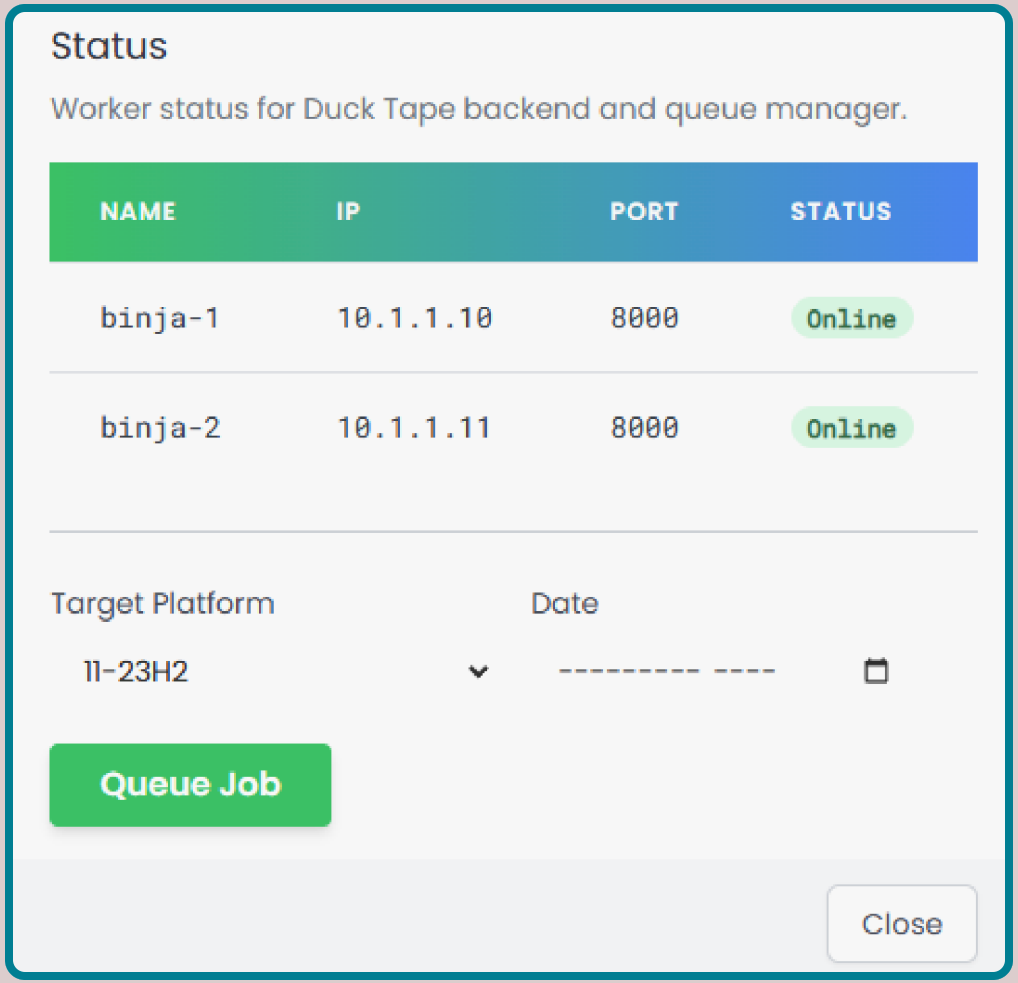
Choose your weapon(s)

- Research requires a toolchain
  - Base tool capability is important (architectures, decompilers, quality)
  - Vulnerability research requires building tools on top of your toolchain, tool choices cascade into your entire pipeline
  - Support for automation, headless processing and API is critical

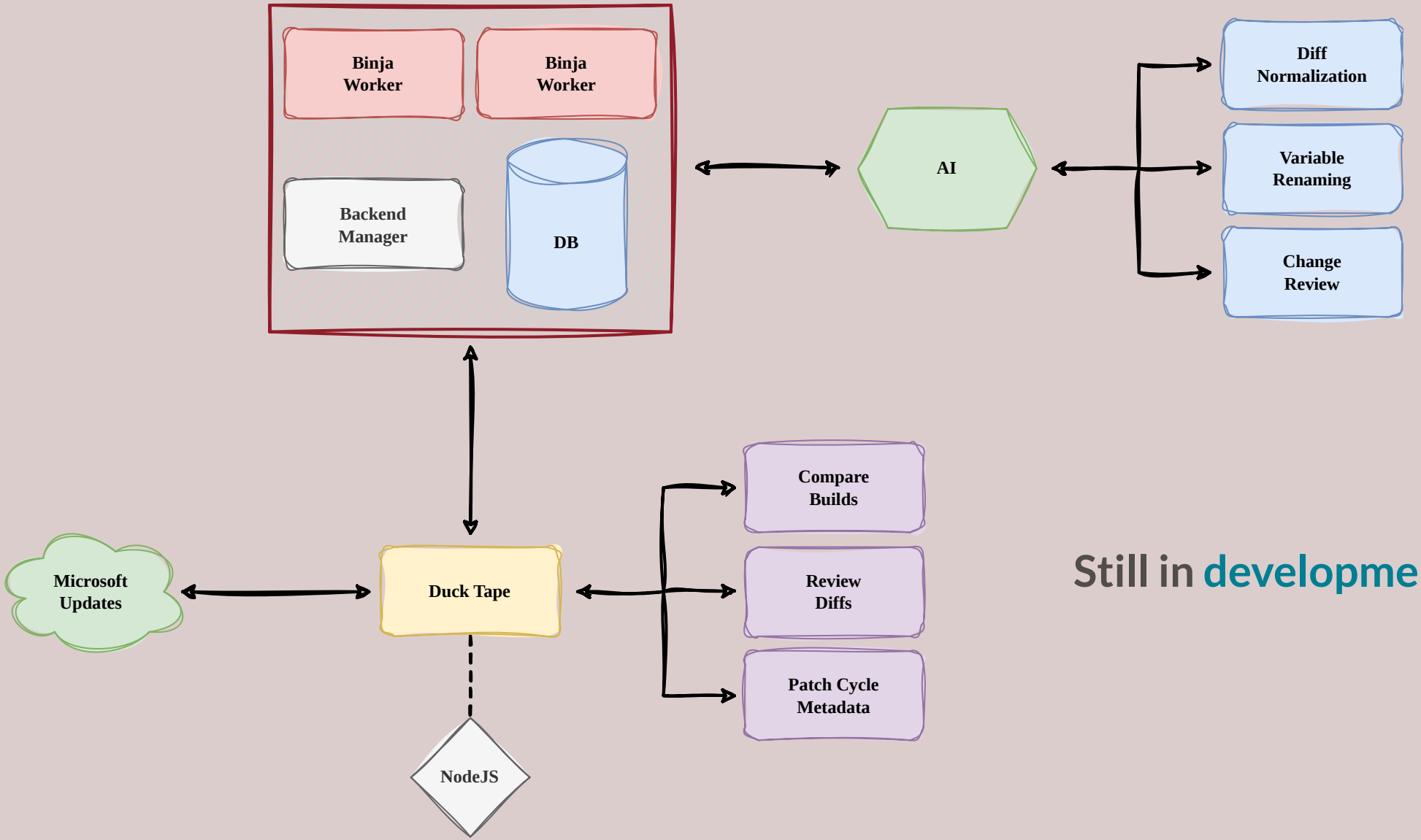
- XOR is primarily a Binary Ninja shop now, with redundancy using Ghidra
  - Vector35 has been a good partner for reporting bugs and extending Binja capabilities
  - Decompiler is good and has been appreciably improving over time
  - API support is excellent
    - https://api.binary.ninja/
    - More API bindings needed (nodeJS & .NET)

Duck Tape

- **What is Duck Tape?**
  - Duck Tape is a **binary diffing engine**
  - **Monaco** editor **code visualization**
  - **Currently** due to patch ingestion it is **focused on Windows** but the diffing **engine generalizes** to other platforms
- **Why do you need a diffing engine?**
  - We want to **reduce cognitive-load**
  - Operating system **patches are vast** and it is not always possible to **track and inspect all changes**
  - **Patch logs don't provide granular information** and also omit changes that have been pushed
  - A lot of inspiration from **@TinySecEx** ♡



Theory Of Design



Still in development!

Diff for tcpip.sys (10.0.22621.3958 -> 10.0.22621.4036)

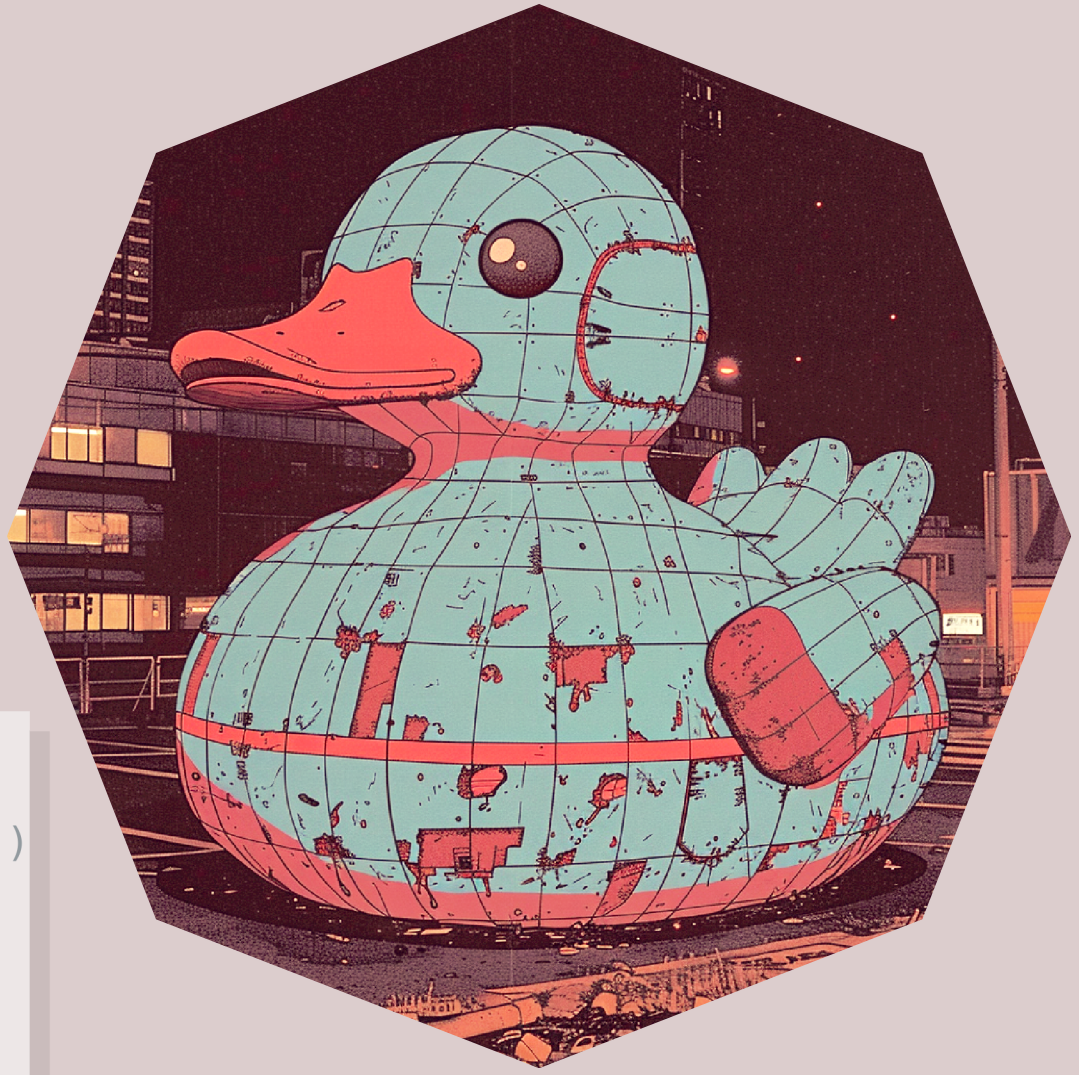
```
183
184     *(arg1 + 0x2c) = 0x3b
185     int32_t var_60_1 = 0
186     var_68.q = 0
187     *(arg1[1] + 0x8c) = 0xc000021b
188
189     if (IppDiscardReceivedPackets(&Ipv6Global, result_1, arg1,
190         int32_t var_58_1
191         var_58_1.b = r14.b
192         var_68.b = r12.b
193         IppSendErrorList(1, &Ipv6Global, arg1, 4, var_68.b, _bs
194
195     result = zx.q(result_1)
196     label_1c006dc78:
197     __security_check_cookie(rax_1 ^ &var_88)
198     return result
```

```
183
184     *(arg1 + 0x2c) = 0x3b
185     int32_t var_60_1 = 0
186     var_68.q = 0
187     *(arg1[1] + 0x8c) = 0xc000021b
188
189     if (IppDiscardReceivedPackets(&Ipv6Global, result_1, arg1,
190         int32_t var_58_1
191         var_58_1.b = r14.b
192         char r9_2 = 4
193         int32_t r10_1
194         r10_1.b = Feature_1223761209__private_IsEnabledDeviceUs
195         char rcx_17 = 1
196         int32_t temp0_3 = _bswap(rsi_1)
197         var_68.b = r12.b
198
199         if (r10_1 == 0)
200             IppSendErrorList(rcx_17, &Ipv6Global, arg1, r9_2, v
201         else
202             IppSendError(rcx_17, &Ipv6Global, arg1, r9_2, var_6
203
204     result = zx.q(result_1)
205     label_1c006dc78:
206     __security_check_cookie(rax_1 ^ &var_88)
207     return result
```

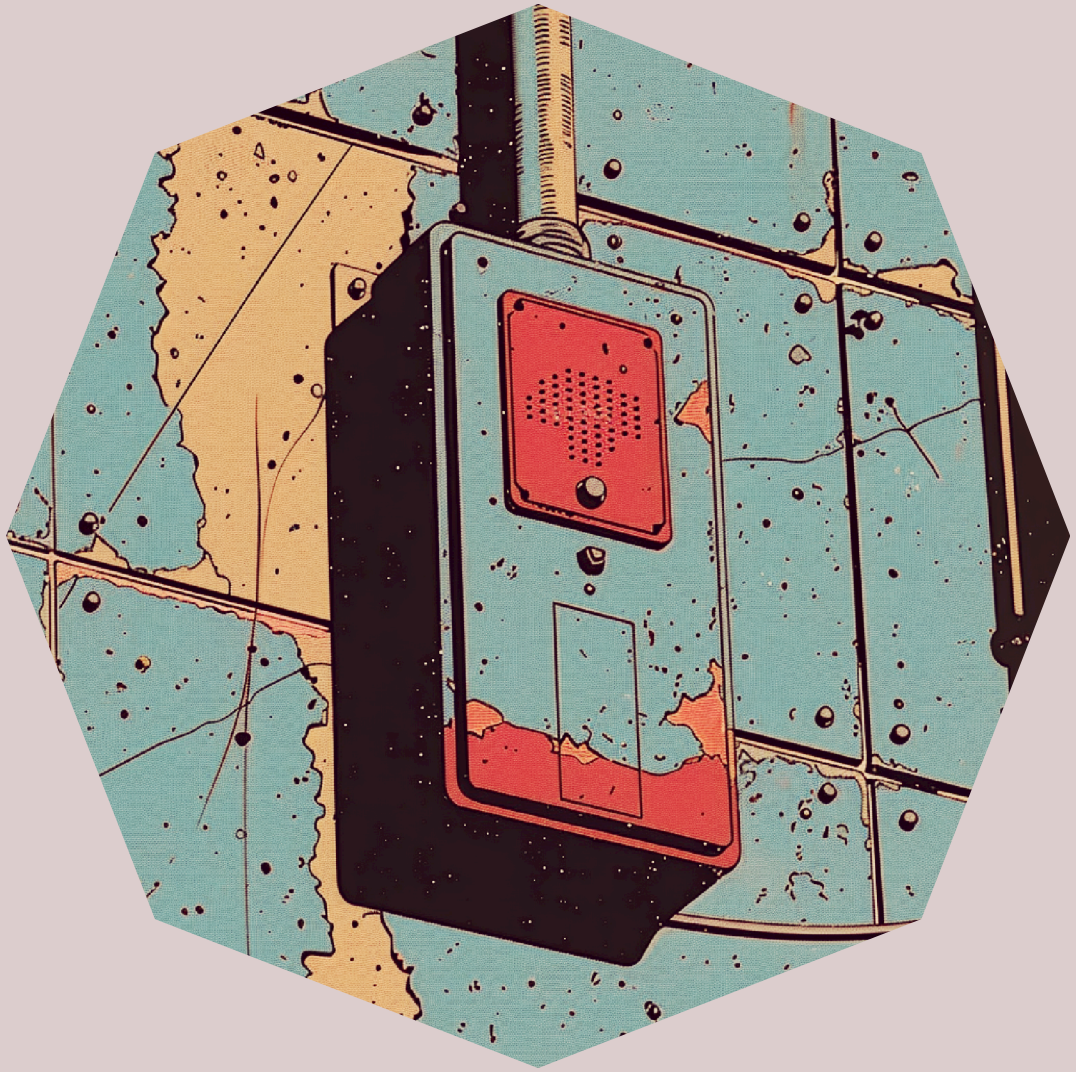
Close

Automating Root Cause Analysis (RCA)

- Programmatically pull and diff patches
- Tell-tale markers for security issues
  - All Microsoft patches are introduced via feature flag
  - Easy to identify
  - Clean condition to use for RCA
    - If Feature Flag enabled, patched code path
    - Else vulnerable code path
- CVE-2024-38063 tcpip RCE (@XiaoWei__ at Kunlun Lab)



```
if
(!Feature_1223761209__private_IsEnabledDeviceUsage())
    IppSendErrorList(rcx_17, &Ipv6Global, arg1,
r9_2, var_68, temp0_3, var_58_1);
else
    IppSendError(rcx_17, &Ipv6Global, arg1, r9_2,
var_68, temp0_3, var_58_1);
```



Trigger Mechanisms

- **Vulnerable code paths chain up from user mode reachable interfaces using program analysis toolkit**
  - **NT Kernel & Win32k** ⇨ system calls
  - **Drivers** ⇨ DispatchDeviceControl, DispatchRead, DispatchWrite, and DispatchReadWrite, DispatchCleanup, DispatchCreate, etc.
  - Select vulnerable function component and **generate condensed backtrace** based on cross-references
    - **Search upwards** till **user entrypoint** is found
    - **Feed** condensed path **to LLM worker** for **POC generation** (maybe)

Case-Study CVE-2024-21338

- From Lazarus (North Korea) with love
- There are actually two bugs here that can be exploited, not one
  - AipAllocateImageContext has an arbitrary increment primitive
  - AppHashComputeImageHashInternal has an arbitrary call primitive

```
2: kd> r
rax=ffffa802f2884ed0 rbx=0000000000000001 rcx=4242424242424242
rdx=ffffa802f2884f98 rsi=4242424242424242 rdi=0000000000000000
rip=fffff80738e7e044 rsp=ffffce8e4425a4e0 rbp=0000000000000000
 r8=0000000000000000  r9=ffffa802f2884f70 r10=fffff80738e7e020
r11=ffffa802f2d27fb8 r12=0000000000000002 r13=0000000000000000
r14=ffffce8e4425a5a0 r15=ffffe08c33569180
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00040246
nt!ObfReferenceObject+0x24:
fffff807`38e7e044 f0480fc15ed0    lock xadd qword ptr [rsi-30h],rbx ds:002b:42424242`42424212
2: kd> k
 # Child-SP          RetAddr               Call Site
00 ffffce8e`4425a4e0 fffff807`419c97dd     nt!ObfReferenceObject+0x24
01 ffffce8e`4425a520 fffff807`419ca4e4     appid!AipAllocateImageContext+0xb9
```

```
Breakpoint 0 hit
appid!AppHashComputeImageHashInternal+0x78:
fffff804`5b53e18c 488b00          mov     rax,qword ptr [rax]
2: kd> r
rax=4343434343434343 rbx=0000000000000000 rcx=ffffe50e8821eb00
rdx=0000000000000000 rsi=0000000000000000 rdi=ffff958de06d15ec
rip=fffff8045b53e18c rsp=ffff958de06d1370 rbp=ffff958de06d1409
 r8=0000000000000001  r9=ffff958de06d14c0 r10=fffff80455166a30
r11=ffff958de06d13a0 r12=0000000000000001 r13=0000000000000000
r14=000000000000000c r15=0000000000000000
iopl=0         nv up ei pl nz na pe nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00040202
appid!AppHashComputeImageHashInternal+0x78:
fffff804`5b53e18c 488b00          mov     rax,qword ptr [rax] ds:002b:43434343`43434343=????????????????
2: kd> u
appid!AppHashComputeImageHashInternal+0x78:
fffff804`5b53e18c 488b00          mov     rax,qword ptr [rax]
fffff804`5b53e18f 488d55ff        lea     rdx,[rbp-1]
fffff804`5b53e193 ff1577b5feff    call    qword ptr [appid!_guard_dispatch_icall_fptr (fffff804`5b529710)]
```

Automate all the things!

But how do we reach these code paths?

```
PS C:\Users\b33f\tools\binja-scripts\symbol-searcher> python.exe .\backtrace.py c:\Users\b33f\tools\ExpDev\CVE-2024-21338-APPID\appid.sys-10.0.22621.2506.b
lob -s AppHashComputeImageHashInternal
[+] Starting backtrace for function: AppHashComputeImageHashInternal (0x1C002E114)

Total input handlers found: 0
Total Call Paths Identified: 4
Total Unique Functions in Call Paths: 19

[+] Call Trace (Shortest Paths First):

AppHashComputeImageHashInternal -> AppHashComputeFileHashesInternal -> AipSmartHashImageFile -> AipDeviceIoControlDispatch -> DriverEntry
    [Input Handlers:
    Address: 0x1C002E114 | Prototype: uint64_t AppHashComputeImageHashInternal(int64_t arg1, int64_t* arg2, int32_t arg3, int64_t* arg4)
    Address: 0x1C0002C94 | Prototype: uint64_t AppHashComputeFileHashesInternal(int64_t arg1, int64_t* arg2, int64_t arg3, uint128_t* arg4, int64_t arg5)
    Address: 0x1C002A46C | Prototype: uint64_t AipSmartHashImageFile(void* arg1, HANDLE arg2, void* arg3, void* arg4)
    Address: 0x1C001DF20 | Prototype: uint64_t AipDeviceIoControlDispatch(int64_t arg1, IRP* arg2)
    Address: 0x1C00350C4 | Prototype: uint64_t DriverEntry(DRIVER_OBJECT* arg1)
    ]
```

```
PS C:\Users\b33f\tools\binja-scripts\symbol-searcher> python.exe .\backtrace.py c:\Users\b33f\tools\ExpDev\CVE-2024-21338-APPID\appid.sys-10.0.22621.2506.b
lob -s AipAllocateImageContext
[+] Starting backtrace for function: AipAllocateImageContext (0x1C0029724)

Total input handlers found: 0
Total Call Paths Identified: 4
Total Unique Functions in Call Paths: 15

[+] Call Trace (Shortest Paths First):

AipAllocateImageContext -> AipSmartHashImageFile -> AipDeviceIoControlDispatch -> DriverEntry
    [Input Handlers:
    Address: 0x1C0029724 | Prototype: uint64_t AipAllocateImageContext(int64_t arg1, HANDLE arg2, int64_t arg3, void** arg4)
    Address: 0x1C002A46C | Prototype: uint64_t AipSmartHashImageFile(void* arg1, HANDLE arg2, void* arg3, void* arg4)
    Address: 0x1C001DF20 | Prototype: uint64_t AipDeviceIoControlDispatch(int64_t arg1, IRP* arg2)
    Address: 0x1C00350C4 | Prototype: uint64_t DriverEntry(DRIVER_OBJECT* arg1)
    ]
```

What about CVE-2024-30088?

- **Exploited** by Emma Kirkpatrick (@carrot_c4k3) at Pwn2Own 2024 in Vancouver
  - Used ITW by APT34 (Iran)
  - TOCTOU leading to arbitrary write using RtlCopyUnicodeString
  - Reachable through NtQueryInformationToken ⇨ TokenAccesInformation

Execution links up to user callable entry point

```
[+] Starting backtrace for function: AuthzBasepCopyoutInternalSecurityAttributes (0x140687EA0)

Total input handlers found: 0
Total Call Paths Identified: 4
Total Unique Functions in Call Paths: 6

[+] Call Trace (Shortest Paths First):

AuthzBasepCopyoutInternalSecurityAttributes -> AuthzBasepQueryInternalSecurityAttributesToken -> SepGetTokenAccessInformationBufferSize -> NtQueryInformationToker
    [Input Handlers:
    Address: 0x140687EA0 | Prototype: uint64_t AuthzBasepCopyoutInternalSecurityAttributes(int32_t* arg1, uint128_t* arg2, int32_t arg3)
    Address: 0x140687D74 | Prototype: int64_t AuthzBasepQueryInternalSecurityAttributesToken(int32_t* arg1, uint128_t* arg2, int32_t arg3, int32_t* arg4)
    Address: 0x140687B78 | Prototype: uint64_t SepGetTokenAccessInformationBufferSize(void* arg1, char arg2, void* arg3, int32_t* arg4, int32_t* arg5, int32_t* ar
g6, int32_t* arg7, int32_t* arg8, int32_t* arg9, int32_t* arg10, int32_t* arg11, int32_t* arg12, int32_t* arg13)
rg3, int32_t* arg4, int32_t* arg5, int32_t* arg6, int32_t* arg7, int32_t* arg8, int32_t* arg9, int32_t* arg10, int32_t* arg11, int32_t* arg12, int32_t* arg13)
    Address: 0x1406EE5A0 | Prototype: uint64_t SeQueryInformationToken(uint128_t* arg1, uint32_t arg2, int32_t* arg3)
    ]

AuthzBasepCopyoutInternalSecurityAttributes -> AuthzBasepQueryInternalSecurityAttributesToken -> SepCopyTokenAccessInformation -> NtQueryInformationToken
    [Input Handlers:
    Address: 0x140687EA0 | Prototype: uint64_t AuthzBasepCopyoutInternalSecurityAttributes(int32_t* arg1, uint128_t* arg2, int32_t arg3)
    Address: 0x140687D74 | Prototype: int64_t AuthzBasepQueryInternalSecurityAttributesToken(int32_t* arg1, uint128_t* arg2, int32_t arg3, int32_t* arg4)
    Address: 0x14068785C | Prototype: uint64_t SepCopyTokenAccessInformation(void* arg1, int32_t** arg2, int32_t arg3, int32_t arg4, int32_t arg5, int32_t arg6, i
nt32_t arg7, int32_t arg8, int32_t arg9, int32_t arg10, int32_t arg11, int32_t arg12, int32_t arg13, char arg14, int64_t* arg15)
    Address: 0x14069B730 | Prototype: int64_t NtQueryInformationToken(uint64_t arg1, uint32_t arg2, char* arg3, int32_t arg4, int32_t* arg5)
    ]
```

LLM automation

- **Smarter models can do smarter workloads**
  - The OpenAI o1 model can actually analyse this backtrace by ingesting function bodies for intermediate calls
  - LLM's with vector stores containing specialized knowledge and RAG can do an ok job at analysis
  - Models can break down tasks into component parts and use function calling in the decompiler to answer their own questions
- **LLM's in VR is a critical area that many people are looking at (quietly)**
  - Project Naptime: Evaluating Offensive Security Capabilities of Large Language Models (Google P0)
  - From Naptime to Big Sleep: Using Large Language Models To Catch Vulnerabilities In Real-World Code (Google P0)
  - Automated LLM Bugfinders (Dave Aitel)

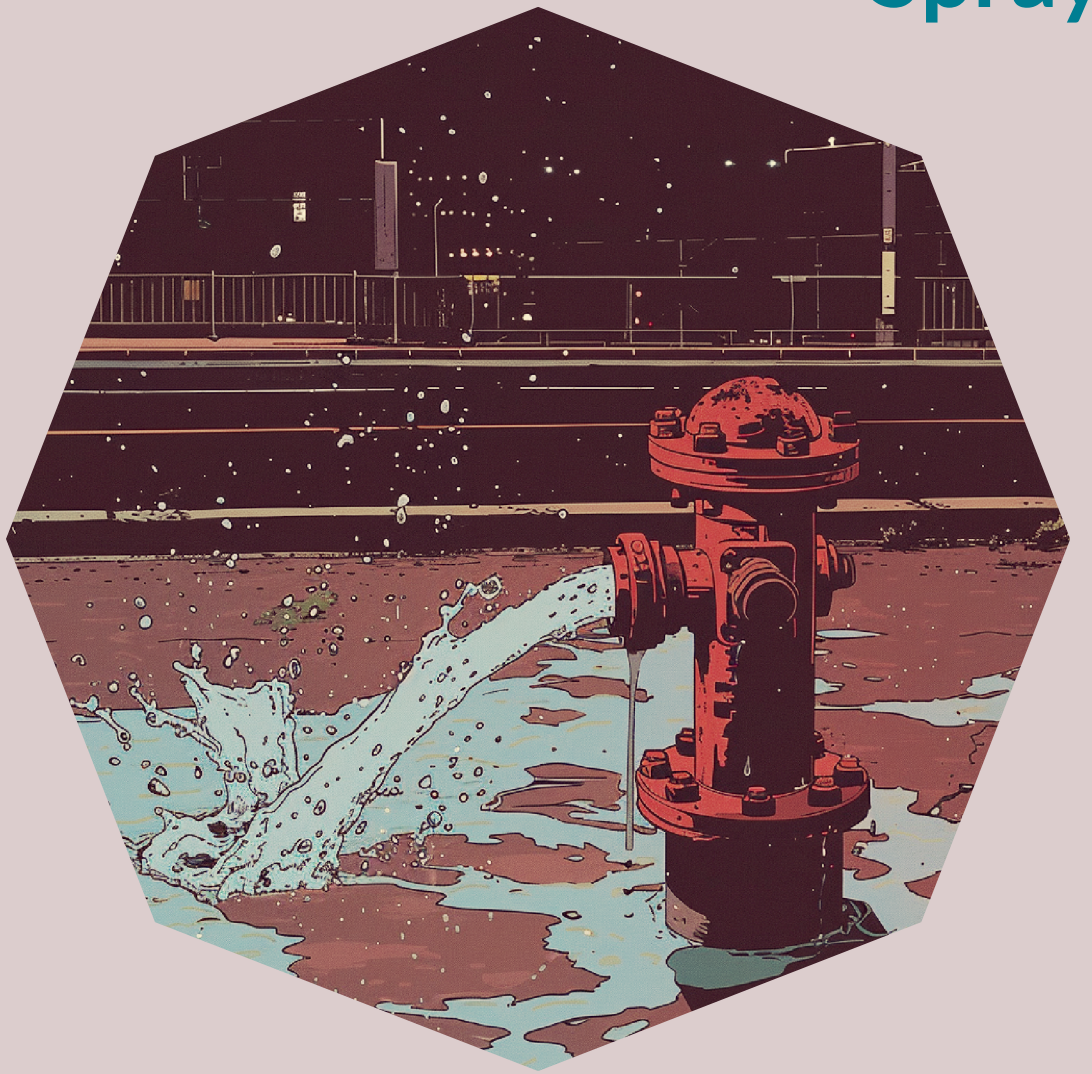
High-level Control Flow (Only Pertinent Steps)

- **NtQueryInformationToken(...)** is invoked with `TokenInformationClass = TokenAccessInformation (0x16)`.
- Inside **NtQueryInformationToken**, it references the token and gathers buffer sizes.
- **SepCopyTokenAccessInformation** is eventually called to build the combined structures.
- **SepCopyTokenAccessInformation** calls **AuthzBasepQueryInternalSecurityAttributesToken** to handle internal security attributes.
- **AuthzBasepQueryInternalSecurityAttributesToken** calls **AuthzBasepCopyoutInternalSecurityAttributes** to output the attributes.
- **AuthzBasepCopyoutInternalSecurityAttributes** invokes **RtlCopyUnicodeString** to copy the attribute's Unicode strings.

Minimal C Code Example (Skeleton)

```c
#include <windows.h>
#include <winternl.h>

// Assume we already have a valid token handle with TOKEN_QUERY access.
// For demonstration, hToken is an already opened token handle.
HANDLE hToken;
DWORD returnLength;
BYTE buffer[1024]; // Sufficiently large buffer

NTSTATUS status = NtQueryInformationToken(
    hToken,
    TokenAccessInformation,      // Trigger the path
    buffer,
    sizeof(buffer),
    &returnLength
);
```

Spraying the Windows Kernel Pool



- Heap/Pool spraying exploitation technique for exploiting several common vulnerability classes
  - Use after free, Double Free, Out-of-Bounds
- Spray objects must match vulnerable object on
  - Paged or NonPaged
  - Size or relative (LFH, Large Pool Alloc, etc)
- Not much research on new objects for pool spraying on Windows
  - Current methods largely meet requirements in *most* cases
  - NonPaged Pool: Named Pipes (buffered/unbuffered)
  - Paged Pool: PipeAttributes, WNF_STATE_DATA
  - Elastic data size and user-controlled data buffer
- Sometimes vulnerability primitives require more work
  - Known spray methods do not apply
  - Influenced/exacerbated by Kernel mitigations & EDR
  - Specialized spray gadget required

Pool Spray Search (Constant Size)

- **Methodology**
  - Search for calls to allocation routines
  - Check **pool flags** match search constraint
  - Check size parameter is **HLIL_CONST**
    - Check if **size** parameter is **within range**
  - Trace call up to match to a user mode **entry point** to ensure allocation can be **triggered from user space**
- **Example gadgets**
  - Searching **ntoskrnl.exe**
  - Searching **POOL_PAGED**
  - Varying max size **32**, **160**, **240**

```
[+] Calling Function -->  NtSetUuidSeed
    |_ HLIL representation:  ExAllocatePool2(0x100, 0x20, 0x64695555)
    |_ Address:  0x140846610
    |_ Pool Tag:  UUid
    |_ Size:  0x20
    |_ Pool Flags:  POOL_FLAG_PAGED

[+] Calling Function -->  NtSetInformationJobObject
    |_ HLIL representation:  ExAllocatePool2(0x100, 0x88, 0x624a7350)
    |_ Address:  0x1407d5c75
    |_ Pool Tag:  PsJb
    |_ Size:  0x88
    |_ Pool Flags:  POOL_FLAG_PAGED

[+] Calling Function -->  NtMapCMFModule
    |_ HLIL representation:  ExAllocatePool2(0x100, 0xf0, 0x636d6650)
    |_ Address:  0x140a01994
    |_ Pool Tag:  Pfmc
    |_ Size:  0xf0
    |_ Pool Flags:  POOL_FLAG_PAGED
```

Pool Spray Search (Elastic Size)

Named Pipes (Buffered)

```
[+] Calling Function -->  NpAddDataQueueEntry
    |_ HLIL representation:  ExAllocatePool2(zx.q((rbx_1 + 0x41).d), zx.q(rcx), 0x7246704e)
    |_ Address:  0x1c000c487
    |_ Pool Tag:  NpFr
    |_ Size:  HLIL_ZX
    [+] Memory copy HLIL representation:  memcpy(&rbx_1[6], UserBuffer, zx.q(arg6))
```

Known primitive

- Methodology
  - Same as constant search, except size is not a constant
  - Process the remainder of the function
    - Memory copy within destination returned by allocation?
    - Possible to tighten constraints (e.g. copy should happen at start of allocation)

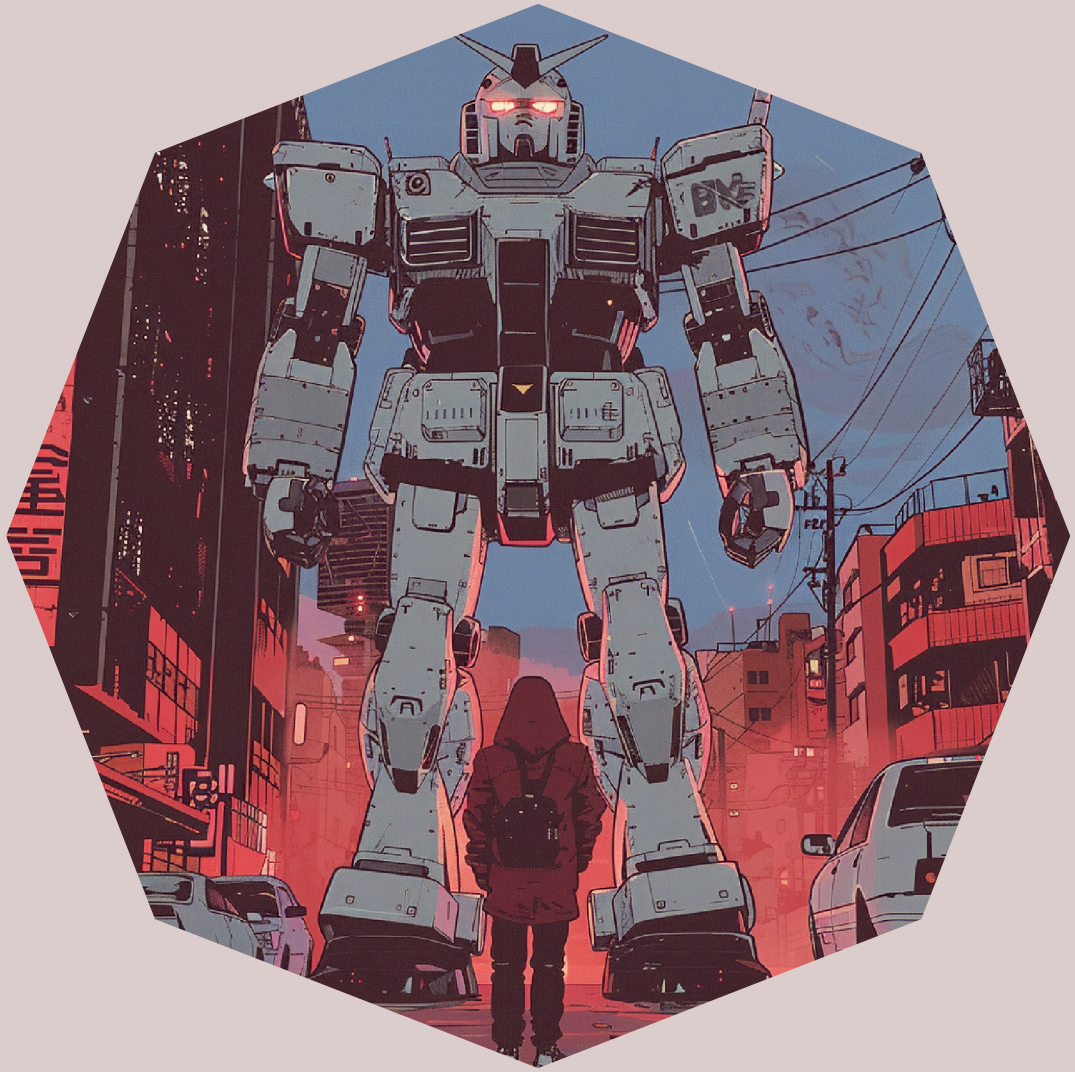
- Easy to find other non-public spray primitives, elastic in size and user controllable data. Some examples:
  - NtSetInformationFile
  - NtSetVolumeInformationFile
  - NtSetEaFile

Unknown primitive

```
[+] Calling Function -->  NtSetInformationFile
    |_ HLIL representation:  ExAllocatePool2(0x61, arg4, 0x42536f49)
    |_ Address:  0x1402f0373
    |_ Pool Tag:  IoSB
    |_ Size:  HLIL_VAR
    |_ Pool Flags:  POOL_FLAG_USE_QUOTA, POOL_FLAG_RAISE_ON_FAILURE, POOL_FLAG_NON_PAGED
    [+] Memory copy HLIL representation:  memcpy(rax_31, arg3, arg4)
```

k(ernel)CFG Primer

- kCFG is an **exploitation mitigation** mechanism in the Windows Kernel that **restricts where execution flow can go**
- Provides protection against techniques like **ROP** - restricts the targets of **indirect calls**
- Like CFG in Windows user mode
- System process has its own **bitmap**, protected by Hypervisor-Protected Code Integrity (**HVCI**)



Mitigations managed

- **Mitigation bypass constraints are often well understood. What is the problem?**
  - Search space is prohibitively large, the entire kernel shares a bitmap!
  - Looking for specialized gadgets manually is impractical even if we know what we are looking for
  - Needle in the haystack problem
- **kCFG and gadget search**
  - From our primitive we know:
    - Argument count
    - What is controlled within the function arguments
    - Desired execution primitive (decrement, increment, ..)
  - Lazarus used an undisclosed kCFG gadget in CVE-2024-21338

kCFG constrained search

Parameter count → HLIL complexity threshold

Call filtering

Semantic Code Analysis

```
PS C:\Users\b33f\tools\binja-scripts\cfg-gadgets> python .\cfg.py C:\Windows\System32\drivers\acpi.sys --number 1 --line_threshold 15
--calls_with_args ExFreePoolWithTag
[+] CFG Metadata
    |_ _load_config_used: 0x1c00600a0
    |_ guardFlags offset: 0x90
    |_ guardFlags: 0x1051c500
    |_ Block Size: 0x5

[>] GFIDS section found at address 0x1c00ad000 with size 3032 bytes.
    |_ Found 606 entries

[+] Entry 523: 0x87540 Absolute Address: 0x1c0087540  Function: ACPIDereferencePnpLocationInterface (1 parameters)
    |_ HLIL Line Count: 9
    |_ Signature: void ACPIDereferencePnpLocationInterface(void* arg1)
    |_ Flag: 00
    |_ Function Body:
1c0087540  void ACPIDereferencePnpLocationInterface(void* arg1)
1c0087546: int32_t temp0 = *(arg1 + 0x30)
1c0087546: *(arg1 + 0x30) -= 1
1c008754d: if (temp0 == 1)
1c008754f: int64_t rax_1 = *(arg1 + 0x20)
1c0087556: if (rax_1 != 0)
1c008755c: rax_1(*(arg1 + 0x10))
1c0087567: ExFreePoolWithTag(arg1, 0)
1c0087578: return
```

kCFG rendered irrelevant

- **Avast did not share the kCFG gadget used by Lazarus**
  - Reason? Avoid "making exploitation too widely accessible"
  - https://decoded.avast.io/janvojtesek/lazarus-and-the-fudmodule-rootkit-beyond-byovd-with-an-admin-to-kernel-zero-day/
- **It doesn't matter: Our method renders kCFG completely ineffective**
  - Quickly search thousands of gadgets with specific requirements for any scenario by querying the semantic structure of valid kCFG targets
  - kCFG payloads can be generated automatically based on obtained primitive
    - trigger vulnerability multiple times to chain calls
    - or use "dispatcher" gadgets
  - Bypass In-The-Wild (ITW) detections
    - Vary exploit chain patterns
    - Switch up call stack

Case Study CVE-2024-30089 Pool Spray

- **Vulnerability exploited by chompie at Pwn2Own 2024 in Vancouver**
  - Logic bug leading to a use after free with a very tight window before object reuse
    - use UAF to overwrite vtable pointer and gain kernel execution
  - Named pipe unbuffered 'IoSB' allocations
    - ExAllocatePoolWithTag (Win 10)
    - ExAllocatePool2 (Win11 - deprecated)

- **Vulnerability complexity**
  - ExAllocatePool2 zero initializes by default
  - Losing race in critical "zero initialized" state will cause NULL pointer dereference and result in BSOD

- **Automation to the rescue**
  - Two options to avoid zero initialization
    - Allocations with POOL_FLAG_UNINITIALIZED
    - Allocations using deprecated API
  - Object needs to be initialized quickly after allocation to have a chance of winning race -- restrict results with direct copies to returned buffer within < 3 LoC (HLIL_VAR_INIT)
  - Search space includes all Kernel modules/drivers reachable from user space, medium integrity
  - Search methodology found an object that met these exact requirements and was used to successfully spray in the tight condition window
    - W32PIDLOCK!win32kbase

Case Study CVE-2024-30089 kCFG

- **Evolving requirements** for gadgets
  - Not all gadgets are made equal!
  - Initial exploit used kCFG gadget within the module of the originating call
    - Second argument **not controllable**
    - **Reliance on volatile register being odd** (reg & 1 == 1) for double-free primitive
    - About **50%** effective before **BSOD**
- **Search scope expanded** to all valid kCFG targets
  - If the **object reference count** is **set to 1**, that guarantees that the object will be freed in vtable release ⇨ **double-free primitive**
  - Using our **targeted search** with some custom conditions resulted in the gadget being found in **about 30 minutes** (VM 8GB RAM)
  - Increased exploit reliability to **100%**

```
# Iterate over the HLIL instructions and increment the line count
try:
    for x in hlil_function.instructions:
        if str(x) == "*(this + 0x18) = 1":
            return True
        if str(x) == "*(arg1 + 0x18) = 1":
            return True

except Exception as e:
    return False
```

```
[+] Entry 228: 0x1a1f0 Absolute Address: 0x1c001a1f0  Function: CTokenBase::Pending (lines: 3)
    |_ Signature: void ?Pending@CTokenBase@@UEAAXXZ(CTokenBase* this)
    |_ HLIL Gadget Found:  *(this + 0x18) = 1
```

Variant analysis

- **We analysed a vulnerability**
  - What happened here?
  - How did this issue become exploitable?
  - What erroneous thought processes did the developer likely follow?

- **We want to investigate, has this happened before**
  - Once we understand the vulnerability we can break it down into its HLIL logical component parts
  - Run a semantic code search on HLIL - like a CodeQL query but for closed source binaries
  - When some combination of features exists elsewhere, we can say that they reproduce the same type of issue

Case-study CVE-2023-21768

- **Root Cause Analysis (RCA)**
  - Introduction of **conditional statement** referring to **PreviousMode value** (KernelMode or UserMode)
  - **Both conditions** execute the **same code**, else condition **missing ProbeForWrite** for UserMode case
  - **Hard to identify** ⇨ small mistake, large function, nondescript variable names

- **Distilling code**
  - Identify all **conditional code** blocks with comparisons for **1** or **0**
  - **Both conditions** execute **identical** code
  - Code block should **assign value** or **call using value argument**
  - Place **length limit** on the conditional **HLIL**

```
1c006fb77 |  |  |  |  if (l.b == 0)
1c006fba4 |  |  |  |      **(ioRequestData + 0x18) = entriesRemoved
1c006fb77 |  |  |  |  else
1c006fb81 |  |  |  |      **(ioRequestData + 0x18) = entriesRemoved
1c006fb81 |  |  |  |
```

```
    [+] Identical blocks found in function AfdNotifyRemoveIoCompletion at address 0x1c006fb77
        |_ Condition: Length.b == 0, Block Length: 0
           |_ True Block:
**(arg3 + 0x18) = var_304
           |_ False Block:
**(arg3 + 0x18) = var_304
```

What about **other Kernel modules**

Talking a closer look

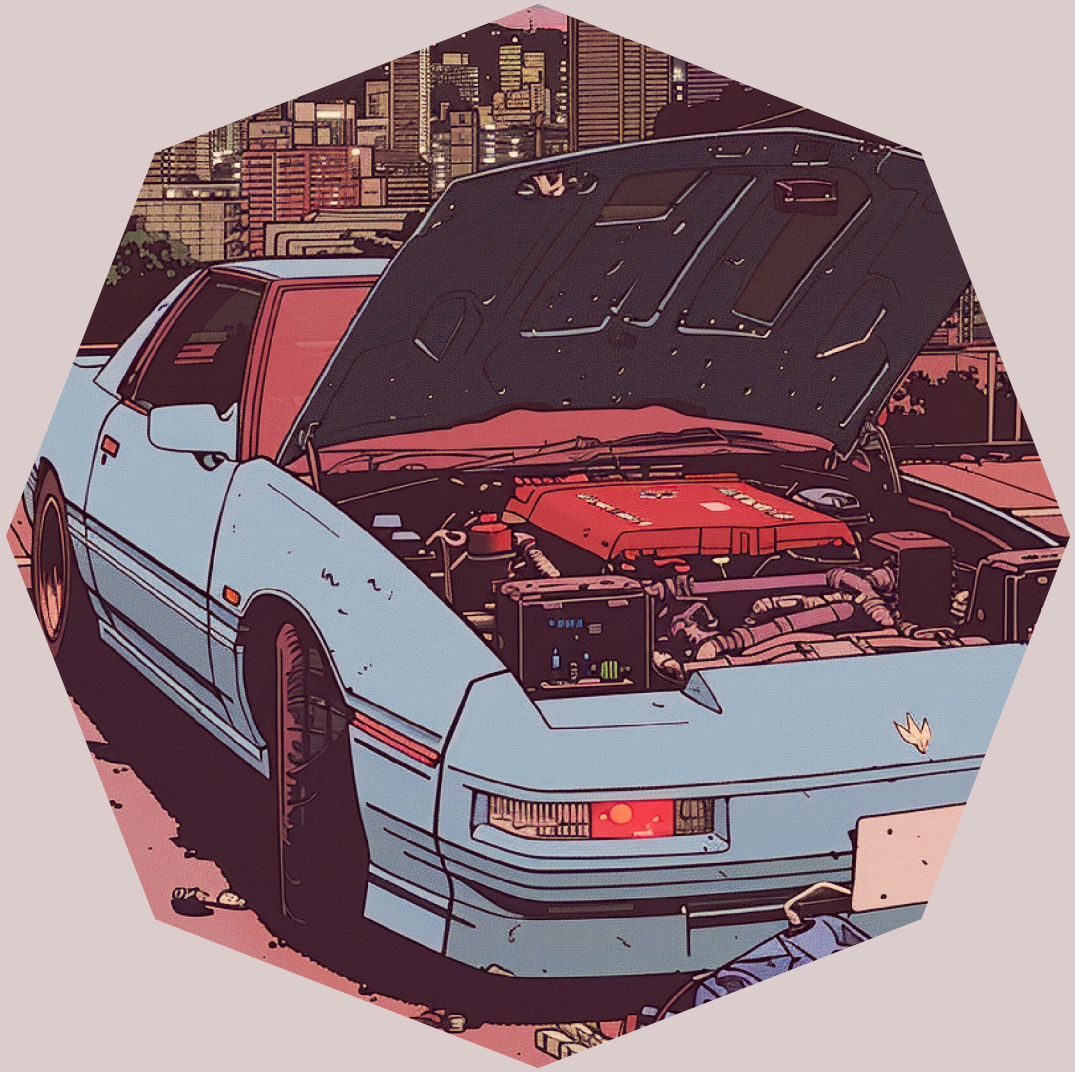
```
[*] Analyzing C:\Windows\System32\drivers\ks.sys...

    [+] Identical blocks found in function SerializePropertySet at address 0x1c00346a8
        |_ Condition: Irp->RequestorMode == 0, Block Length: 1
            |_ True Block:
                memcpy(rax_2, rdx_2, r14_1)
            |_ False Block:
                memcpy(rax_2, rdx_2, r14_1)

    [+] Identical blocks found in function UnserializePropertySet at address 0x1c0034986
        |_ Condition: Irp->RequestorMode == 0, Block Length: 1
            |_ True Block:
                memcpy(rax_2, rdx_1, r15_1)
            |_ False Block:
                memcpy(rax_2, rdx_1, r15_1)
```

```
PVOID pType3InputBuffer = *(uint64_t*)((char*)pStack->Parameters + 0x18);

if (!pIrp->RequestorMode)
    memcpy(P, pType3InputBuffer, NumberOfBytes);
else
    memcpy(P, pType3InputBuffer, NumberOfBytes);
```

TOCTOU ⇨ ks.sys

- A system-wide variant scan returned results
  - We took a closer look at some of the findings, including this one
  - At the time we, *incorrectly*, evaluated this as non-exploitable 😢
  - Later this was assigned CVE-2024-30084

- This was exploited by Devcore at Pwn2Own 2024 in Vancouver
  - Angelboy (@scwuaptx) combined CVE-2024-30084 and CVE-2024-35250 to achieve LPE
  - https://devco.re/blog/2024/08/23/streaming-vulnerabilities-from-windows-kernel-proxying-to-kernel-part1-en/



Maintenance is ~~not~~ sexy

- **Maintenance is not super exciting**
  - The exploit **works on my machine**
  - The exploit **doesn't crash my target**
  - The exploit **targets my version**
- **Compatibility and stability are essential**
  - We really **want to do cool hacking shit** but it is important to remember that **the product has to be useful**
  - Maintenance covers many areas like VM automation, automated application deployment and conditional code updates
  - Common code update tasks include **symbol resolution** and **gadget portability**

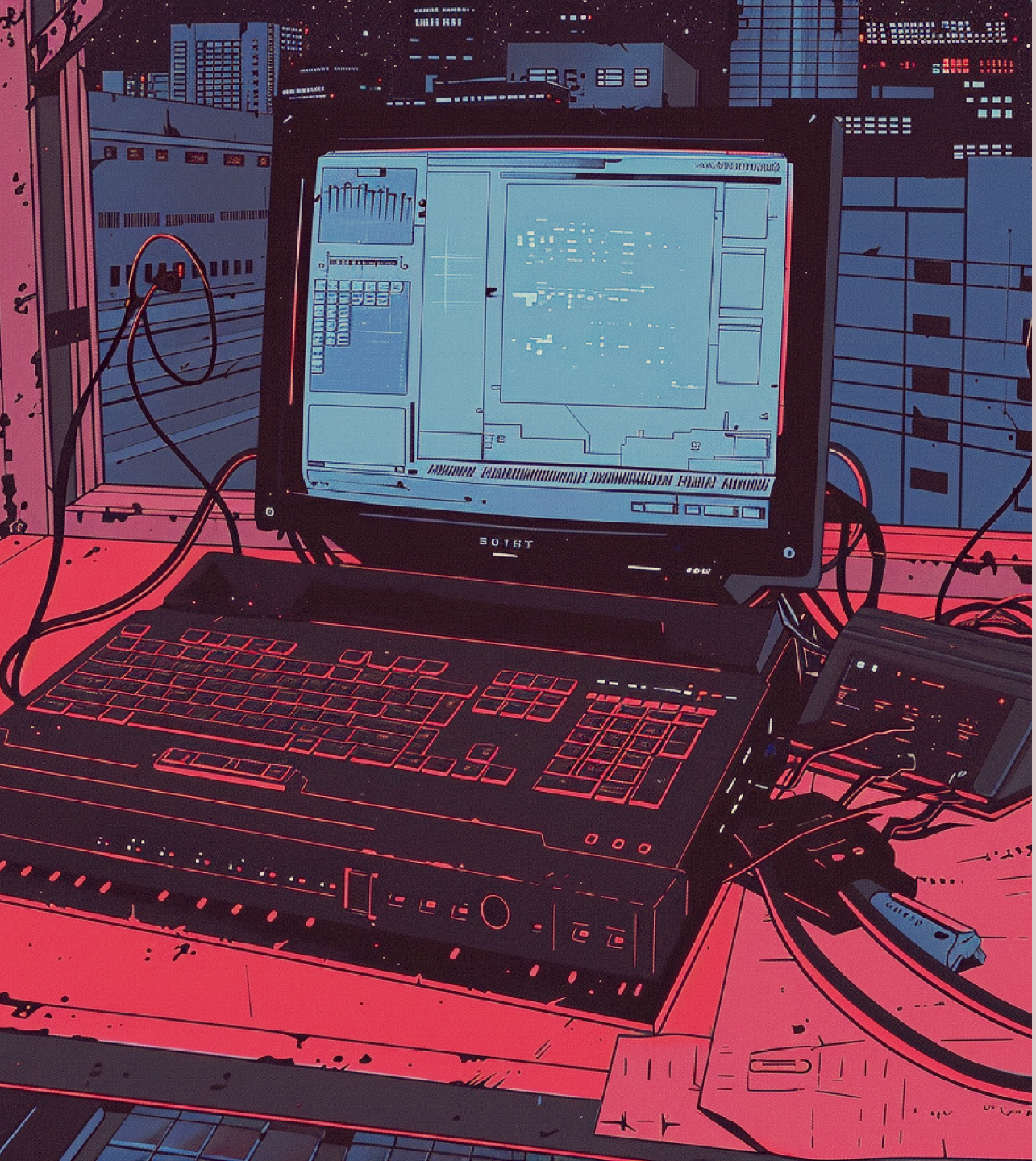
Symbol and structure resolution

```
PS C:\Users\b33f\tools\binja-scripts\symbol-searcher> python .\search-symbol.py C:\Windows\System32\drivers\acpi
-p SOURCE_DESCRIPTOR

[!] No symbols matched the partial name.

[+] Matched Types:
+---------------------------------------+-----------------------------------------------+
| Type Name                             | Type Definition                               |
+=======================================+===============================================+
| CM_FULL_RESOURCE_DESCRIPTOR           | struct _CM_FULL_RESOURCE_DESCRIPTOR           |
+---------------------------------------+-----------------------------------------------+
| CM_PARTIAL_RESOURCE_DESCRIPTOR        | struct _CM_PARTIAL_RESOURCE_DESCRIPTOR        |
+---------------------------------------+-----------------------------------------------+
| IO_RESOURCE_DESCRIPTOR                | struct _IO_RESOURCE_DESCRIPTOR                |
+---------------------------------------+-----------------------------------------------+
| PCM_FULL_RESOURCE_DESCRIPTOR          | struct _CM_FULL_RESOURCE_DESCRIPTOR*          |
+---------------------------------------+-----------------------------------------------+
| PCM_PARTIAL_RESOURCE_DESCRIPTOR       | struct _CM_PARTIAL_RESOURCE_DESCRIPTOR*       |
+---------------------------------------+-----------------------------------------------+
| PHYSICAL_COUNTER_RESOURCE_DESCRIPTOR  | struct _PHYSICAL_COUNTER_RESOURCE_DESCRIPTOR  |
+---------------------------------------+-----------------------------------------------+
| PHYSICAL_COUNTER_RESOURCE_DESCRIPTOR_TYPE | enum _PHYSICAL_COUNTER_RESOURCE_DESCRIPTOR_TYPE |
+---------------------------------------+-----------------------------------------------+
| PIO_RESOURCE_DESCRIPTOR               | struct _IO_RESOURCE_DESCRIPTOR*               |
+---------------------------------------+-----------------------------------------------+
```

```
[+] Searching for unique symbol 'AcpiWrapperReadConfig'

Symbol Type: FunctionSymbol
Symbol Address: 0x1C0001810
Symbol Name: AcpiWrapperReadConfig
Function Name: AcpiWrapperReadConfig
Return Type: int64_t
Arguments:
  Arg 1: int32_t arg1
  Arg 2: int32_t arg2
```

```
[+] Searching for unique type '_WHEA_ERROR_SEVERITY':

Type Name: _WHEA_ERROR_SEVERITY
Type Definition: Enumeration (int32_t)
Enumerators:
+-------------------------+---------+
| Enumerator              | Value   |
+=========================+=========+
| WheaErrSevRecoverable   | 0x0     |
+-------------------------+---------+
| WheaErrSevFatal         | 0x1     |
+-------------------------+---------+
| WheaErrSevCorrected     | 0x2     |
+-------------------------+---------+
| WheaErrSevInformational | 0x3     |
+-------------------------+---------+
```

```
[+] Searching for unique type '_WHEA_ERROR_RECORD_HEADER':

Type Name: _WHEA_ERROR_RECORD_HEADER
Type Definition: Structure
Members:
+----------+-----------------------------------+-----------------+
| Offset   | Type                              | Name            |
+==========+===================================+=================+
| 0x0      | ULONG                             | Signature       |
+----------+-----------------------------------+-----------------+
| 0x4      | WHEA_REVISION                     | Revision        |
+----------+-----------------------------------+-----------------+
| 0x8      | ULONG                             | SignatureEnd    |
+----------+-----------------------------------+-----------------+
| 0xC      | USHORT                            | SectionCount    |
+----------+-----------------------------------+-----------------+
| 0x10     | WHEA_ERROR_SEVERITY               | Severity        |
+----------+-----------------------------------+-----------------+
| 0x14     | WHEA_ERROR_RECORD_HEADER_VALIDBITS | ValidBits       |
+----------+-----------------------------------+-----------------+
| 0x18     | ULONG                             | Length          |
+----------+-----------------------------------+-----------------+
| 0x20     | WHEA_TIMESTAMP                    | Timestamp       |
+----------+-----------------------------------+-----------------+
| 0x28     | GUID                              | PlatformId      |
+----------+-----------------------------------+-----------------+
| 0x38     | GUID                              | PartitionId     |
+----------+-----------------------------------+-----------------+
| 0x48     | GUID                              | CreatorId       |
+----------+-----------------------------------+-----------------+
| 0x58     | GUID                              | NotifyType      |
+----------+-----------------------------------+-----------------+
| 0x68     | ULONGLONG                         | RecordId        |
+----------+-----------------------------------+-----------------+
| 0x70     | WHEA_ERROR_RECORD_HEADER_FLAGS    | Flags           |
+----------+-----------------------------------+-----------------+
| 0x78     | WHEA_PERSISTENCE_INFO             | PersistenceInfo |
+----------+-----------------------------------+-----------------+
| 0x80     | UCHAR[0xc]                        | Reserved        |
+----------+-----------------------------------+-----------------+
```



Conclusions and Future-Work

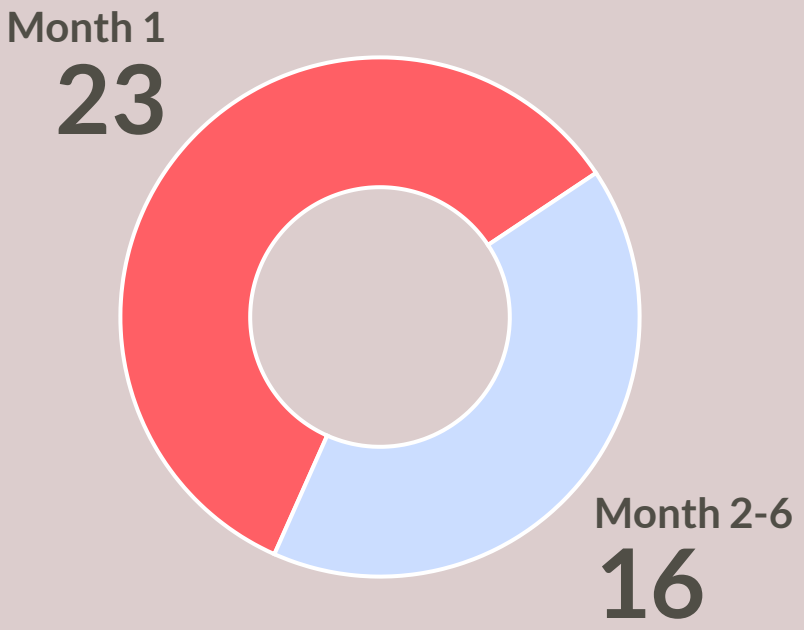
Life on the loop

Time-to-Exploit Trends

Most exploitation happens in the first month after disclosure, data from Mandiant

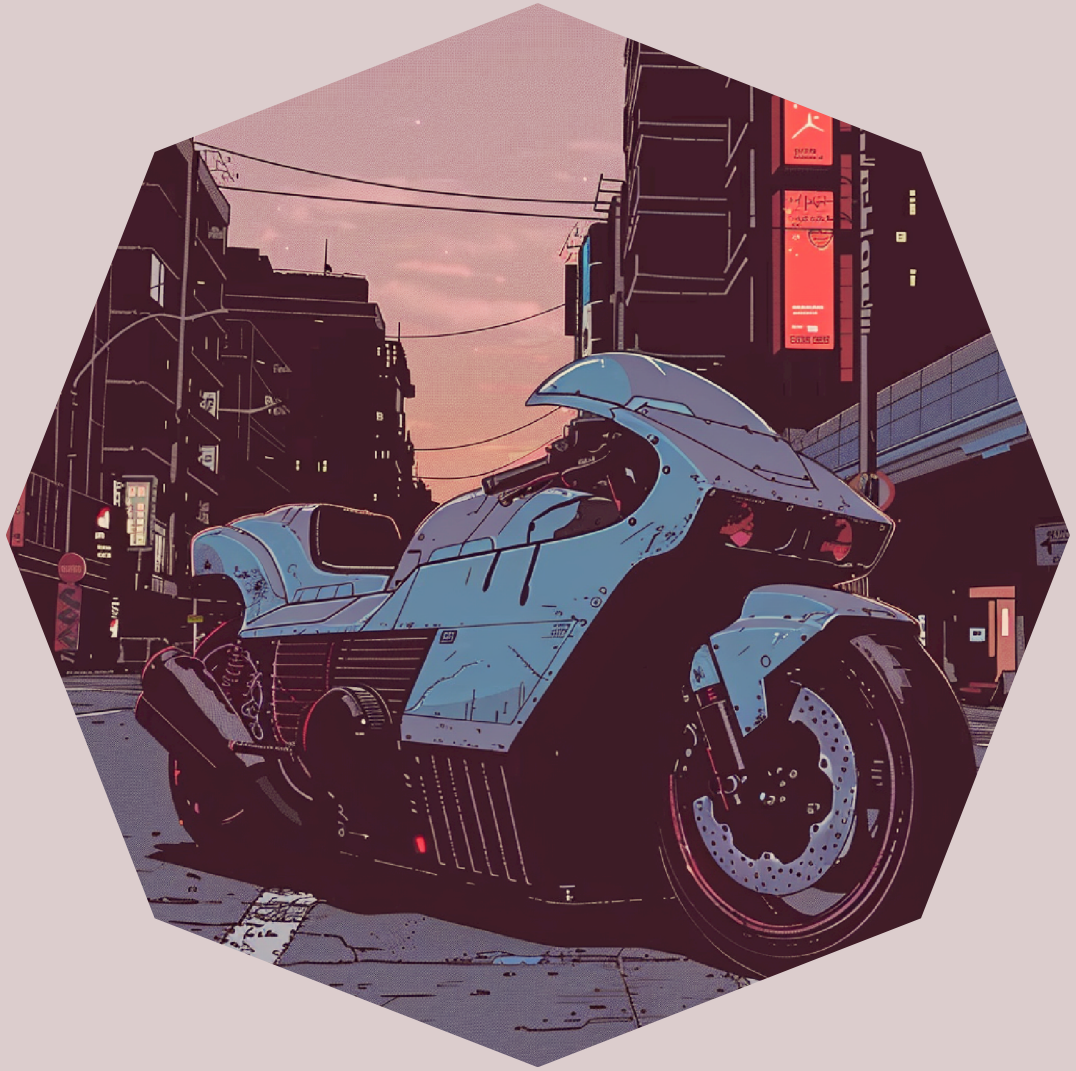
Breakdown by Month

Total Distribution



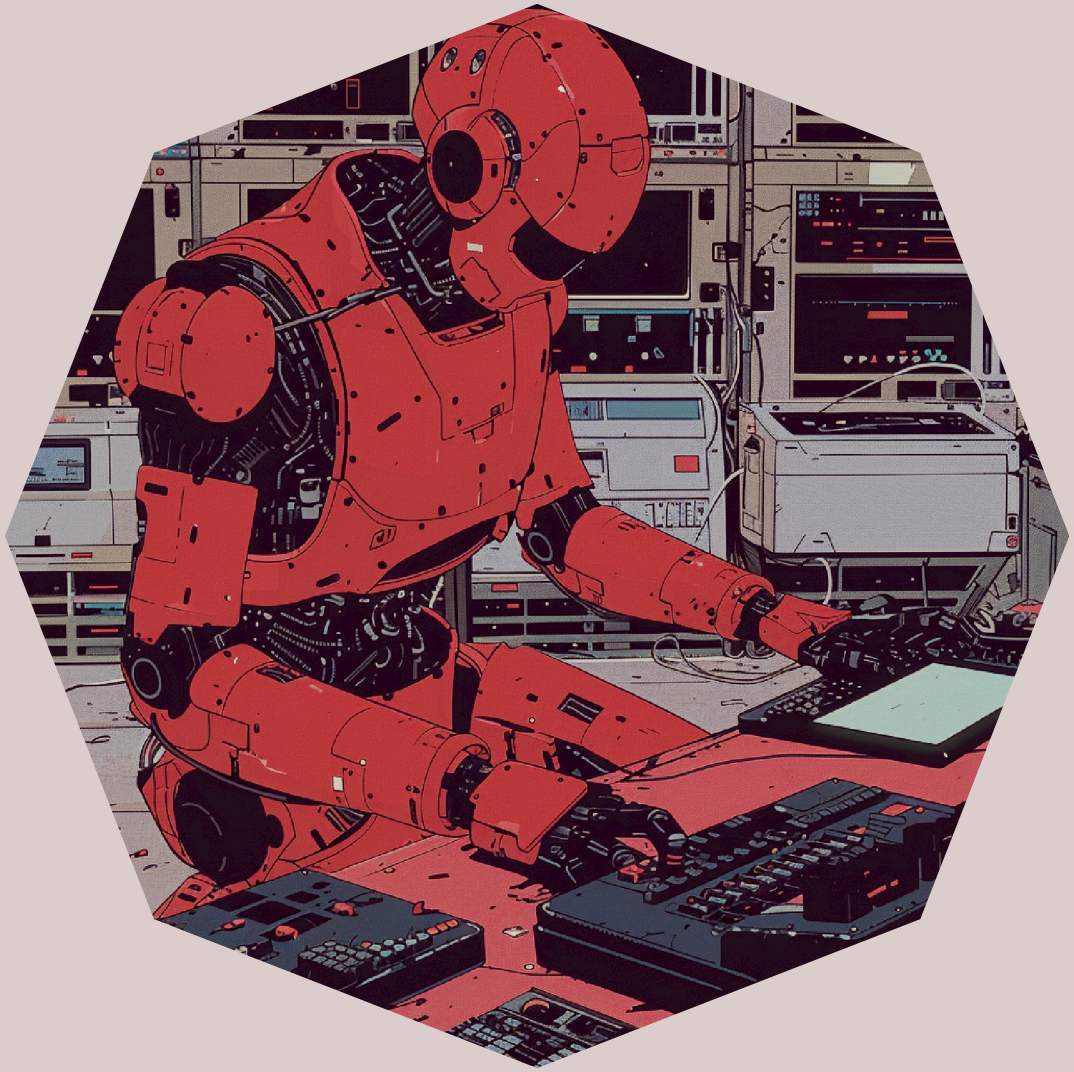
If it ain't broke... still fix it!

- Many compelling reasons to build a library of swappable exploit objects & primitives
  - In-The-Wild (ITW) detections
  - Even commercial EDRs can detect mainstream exploitation methods
    - CrowdStrike knows about your pipe sprays 💀
- Staying ahead of the curve reduces future work
  - Future mitigations will continue to make exploitation harder
  - Beneficial to have multiple back up methods that do the same thing



Conclusions

- **Vulnerability research and exploit development is hard work**
  - Even though suffering is a requirement, **you shouldn't suffer needlessly**
  - **Complex** and **deterministic** tasks should be **automated** to reduce manual labour
- **Training AI agents**
  - Training specialized agents is key, **layer specialized knowledge** on top of **frontier models**
  - Models are already very smart for general use cases but not yet in for this specialized use case
  - Model "**smartness**" is the **only bottleneck that matters**
  - **AI** will play a crucial role in **NatSec** and **VR/RE**
- **Program Analysis should power automated VR and be the backbone of AI reasoning**



Problem statements

- **Train LLM's to reason better about Root Cause Analysis (RCA) and PoC generation**
  - Multi-agent workflow
    - Planner, Reasoner, Questioner
- **Future Workflow Improvements**
  - Allocation search doesn't take into account object persistence
    - Can try to solve by looking for free calls in non failure paths
  - Finetuning, RAG and vector storage for special characteristics combined with extensive tool use (e.g. SMT solvers)
    - Models fail on complex scenarios and require hand-holding, partially engineering problem, partially fundamental "smartness" problem
    - Reaching vulnerable code doesn't mean the vulnerability is triggered, specific conditions must be met

Questions?

IBM®